数据库系统概念知识点笔记

第1章 数据库基本概念

数据库管理系统（database management system, DBMS）是由一个**相互关联的数据的集合**（数据库）和一组用以**访问这些数据的程序**组成；

设计系统的目的：得到一个可以方便、高效地存取数据库信息的环境，用该系统管理大量信息；

与之对应的文件系统的缺点：数据冗余和数据不一致性、获取数据困难、数据的孤立：数据分散在各个文件中、完整性问题：原子性问题、并发访问异常、安全性问题；

数据抽象的3个层次：物理层、逻辑层、视图层；

物理数据独立性：虽然逻辑层的简单结构的实现可能涉及复杂的物理层结构，但逻辑层的用户不必知道这样的复杂性（（应用程序不依赖于物理模式）；

数据库实例：特定时刻存储在数据库中的信息的集合称作数据库的一个实例（给定时刻数据库中数据的一个快照）。

数据库模式：数据库的总体设计。有物理模式、逻辑模式、子模式等（数据库的逻辑设计）。

数据模型：数据库结构的基础是数据模型。数据模型是一个描述数据、数据联系、数据语义以及一致性约束的概念工具的集合。描述物理层、逻辑层、视图层的数据库设计方式。

数据模型的分类：关系模型、实体-联系模型、基于对象数据模型、半结构化数据模型；

数据库提供两种语言：一种是**数据定义语言**（data-definition language, DDL, 定义数据库的模式），另一个是**数据操纵语言**（data-manipulation language, DML, 用于表达数据库的查询和更新）。

两类基本的数据操纵语言：过程化DML、声明式DML。

一致性约束：简称约束，存储在数据库中的数据值必须满足的一些一致性约束；

域约束：每个属性都必须对应于一个所有可能的取值构成的域；

参照完整性：一个关系中给定属性集上的取值也在另一关系的某一属性集的取值中出现；

断言：数据库需要时刻满足的某一条件（域约束和参照完整性是断言的特殊形式）；

授权：对于不同的用户在数据库中的不同数据值上允许不同的访问类型（读权限、插入权限、更新权限、删除权限）；

数据字典：可视作一种特殊的表，该表只能由数据库系统本身来访问和修改，包含元数据。

元数据：数据的数据。

第2章 关系数据库

关系数据库：基于关系模型，使用一系列表（该种数据库是由表的集合构成的）来表达数据以及这些数据之间的联系；

关系——表、**元组**——行、**属性**——列（**域**——每个属性有的一组允许的值）；

关系实例：表示一个关系的特定实例（一组特定的行）；

关系是元组集合，具有无序性；

原子域、非原子域；

null（空值）是一个特殊的值，表示值不存在或者不知道。

关系的概念对应于程设中变量的概念，**关系模式**的概念对应于程设中类型定义的概念；

码（key）

- 超码(superkey)：一个或多属性集合，可在一个关系中唯一地标识一个元组。
- 候选码(candidate key)：任意真子集都不能成为超码的超码。
- 主码(primary key)：被设计者选中，用来区分不同元组的候选码。
- 外码(foreign key)：一个关系模式(r1)在它的属性中包含的另一个关系模式(r2)的主码，该属性在r1上被称为参照r2的**外码**。关系r1被称为外码依赖的**参照关系**，r2叫做外码的**被参照关系**。

模式图：是数据库中模式的图形化表示，它显示了数据库中的关系、关系的属性、主码和外码。

关系查询语言：定义了一组运算集，这些运算可作用于表上，并输出表作为结果，这些运算可以组合成表达式，表达所需的查询。

关系代数提供了一组运算，它们以一个或者多个关系作为输入，返回一个关系作为输出。

第3章 SQL

SQL语言有以下几个部分：数据定义语言，数据操纵语言，完整性，视图定义，事务控制，嵌入式SQL和动态SQL，授权；

SQL的一些命令和关键字：creat table, primary key, foreign key, reference, not null, insert, delete, drop table, alter, select,；

自然连接：作用于两个关系，并产生一个关系作为结果（只考虑那些在两个关系模式中都出现的属性上取值相同的元组对）；

集合运算：union、intersect、except；

后面基本是一些运算方式，跳过；

第4章 中级SQL

外连接（outer join)运算与之前的连接运算类似，但通过在结果中创建包含**空值的元组**的方式，**保留了那些在连接中丢失的元组**。分为左外连接、右外连接、全外连接三种。

视图：为了保护数据，避免逻辑层直接暴露给用户，创建视图来解决。不是逻辑模型的一部分，但作为虚关系对用户可见的关系。

物化视图：如果定义视图的实际关系发生改变，视图也会跟着修改。保持物化视图一直在最新状态的过程称为**物化视图维护**，或简称**视图维护**。

不是所有的视图都可以更新。

事务由**查询**和（或）**更新的语句**的序列组成。SQL标准规定当一条SQL语句执行时，就隐式地开始了一个事务。

事务具有：原子性、隔离性、一致性、持久性。

参照完整性约束保证**授权用户**对数据库所作的**修改**不会破坏**数据的一致性**（参见第一章）。

对数据的授权包含：授权读取数据、授权插入新数据、授权更新数据、授权删除数据。

每种类型的授权都称为一个**权限**。

除了对数据的授权，还可以在**数据库模式**上的授权。最大的授权形式是被授予**数据库管理员**（超级用户）的。

权限的授权和收回、角色在授权方面的作用、视图和模式的授权、权限的转移。

课本的总结：

- SQL 支持包括内连接、外连接在内的几种连接类型，以及几种形式的连接条件。
- 视图关系可以定义为包含查询结果的关系。视图是有用的，它可以隐藏不需要的信息，可以把信息从多个关系收集到一个单一的视图中。
- 事务是一个查询和更新的序列，它们共同执行某项任务。事务可以被提交或回滚。当一个事务被回滚，该事务执行的所有更新所带来的影响将被撤销。
- 完整性约束保证授权用户对数据库所做的改变不会导致数据一致性的破坏。
- 参照完整性约束保证出现在一个关系的给定属性集上的值同样出现在另一个关系的特定属性集上。
- 域约束指定了在一个属性上可能取值的集合。这种约束也可以禁止在特定属性上使用空值。
- 断言是描述性表达式，它指定了我们要求总是为真的谓词。
- SQL 数据定义语言提供对定义诸如 **date** 和 **time** 那样的固有域类型以及用户定义域类型的支持。
- 通过 SQL 授权机制，可以按照在数据库中不同数据值上数据库用户所允许的访问类型对他们进行区分。
- 获得了某种形式授权的用户可能允许将此授权传递给其他用户。但是，对于权限怎样在用户间传递我们必须很小心，以保证这样的权限在将来的某个时候可以被收回。
- 角色有助于根据用户在组织机构中所扮演的角色，把一组权限分配给用户。

第6章 形式化关系查询

课本总结：

- **关系代数**(relational algebra)定义了一套在表上运算且输出结果也是表的代数运算。这些运算可以混合使用来得到表达所希望查询的表达式。关系代数定义了关系查询语言中使用的基本运算。
- 关系代数运算可以分为:
  - □ 基本运算。
  - □ 附加的运算,可以用基本运算表达。
  - □ 扩展的运算,其中的一些扩展了关系代数的表达能力。
- 关系代数是一种简洁的、形式化的语言,不适合于那些偶尔使用数据库系统的用户。因此,商用数据库系统采用有更多"语法修饰"的语言。从第 3 章到第 5 章我们讨论了最有影响力的语言——**SQL**,它是基于关系代数的。
- **元组关系演算**(tuple relational calculus)和**域关系演算**(domain relational calculus)是非过程化语言,代表了关系查询语言所需的基本能力。基本关系代数是一种过程化语言,在能力上等价于被限制在安全表达式范围内的关系演算的这两种形式。
- 关系演算是简洁的、形式化的语言,并不适合于那些偶尔使用数据库系统的用户。这两种形式化语言构成了两种更易使用的语言 QBE 和 Datalog 的基础,我们将在附录 B 中介绍它们。

元组关系公式P的**域**:P所引用的所有值的集合。

第7章 数据库设计和E-R模型

- 数据库设计主要涉及数据库模式的设计。**实体 - 联系**(Entity-Relationship,E-R)数据模型是一个广泛用于数据库设计的数据模型。它提供了一个方便的图形化表示方法以查看数据、联系和约束。
- E-R 模型主要用于数据库设计过程。它的发展是为了帮助数据库设计,这是通过允许定义**企业模式**(enterprise schema)实现的。这种企业模式代表数据库的全局逻辑结构,该全局结构可以用 **E-R 图**(E-R diagram)图形化地表示。
- **实体**(entity)是在现实世界中存在并且区别于其他对象的对象。我们通过把每个实体同描述该实体的一组属性相关联来表示区别。
- **联系**(relationship)是多个实体间的关联。相同类型的联系的集合为**联系集**(relationship set),相同类型的实体的集合为**实体集**(entity set)。
- 术语**超码**(superkey)、**候选码**(candidate key)以及**主码**(primary key)同适用于关系模式一样适用于实体和联系集。在确定一个联系集的主码时需要小心,因为它由来自一个或多个相关的实体集的属性组成。
- **映射的基数**(mapping cardinality)表示通过联系集可以和另一实体相关联的实体的个数。
- 不具有足够属性构成主码的实体集称为**弱实体集**(weak entity set)。具有主码的实体集称为**强实体集**(strong entity set)。
- E-R 模型的各种性质为数据库设计者提供了大量的选择,使设计人员可以最好地表示被建模的企业。在某些情况中,概念和对象可以用实体、联系或属性来表示。企业总体结构的各方面可以用弱实体集、概化、特化或聚集很好地描述。设计者通常需要在简单的、紧凑的模型与更精确但也更复杂的模型之间进行权衡。

- 用 E-R 图定义的数据库设计可以用关系模式的集合来表示。数据库的每个实体集和联系集都有唯一的关系模式与之对应,其名称即为相应的实体集或联系集的名称。这是从 E-R 图转换为关系数据库设计的基础。
- **特化**(specialization)和**概化**(generalization)定义了一个高层实体集和一个或多个低层实体集之间的包含关系。特化是取出高层实体集的一个子集来形成一个低层实体集。概化是用两个或多个不相交的(低层)实体集的并集形成一个高层实体集。高层实体集的属性被低层实体集继承。
- **聚集**(aggregation)是一种抽象,其中联系集(和跟它们相关的实体集一起)被看作高层实体集,并且可以参与联系。
- UML 是一种常用的建模语言。UML 类图广泛用于对类建模以及一般的数据建模。

实体通过一组**属性**来表示,**属性**是实体集中每个成员所拥有的描述性性质。

参与联系集的实体集的数目称为联系集的**度**。

简单/复合属性:简单属性不能划分为更小的部分,复合属性则可以。

单值/多值属性：单值（属性对一个特定实体都只有单独的一个值）、多值（一个属性可能对应一组值）。

派生属性：这类属性的值可以从别的相关属性或实体派生出来。

映射基数有以下种情况：一对一、一对多、多对一、多对多。

弱实体集必须与另一个称作**标识**或**属主实体集**的实体集关联才能有意义。弱实体集**存在依赖**于标识实体集。标识实体集**拥有**它所标识的弱实体集。将弱实体集与其标识实体集相联的联系称为**标识性联系**。

弱实体集的**分辨符**：起到类似主码的作用。

重叠特化：一个实体集可能属于多个特化实体集（反之则是**不相交特化**）。

高层与低层实体集可以分别称作**超类**与**子类**。

第8章 关系数据库设计

- **我们给出了数据库设计中易犯的错误，和怎样系统地设计数据库模式来避免这些错误。这些错误包括信息重复和不能表示某些信息。**
- **我们给出了从 E-R 设计到关系数据库设计的发展过程，什么时候可以安全地合并模式，什么时候应该分解模式。所有有效的分解都必须是无损的。**
- **我们描述了原子域和第一范式的假设。**
- **我们介绍了函数依赖的概念，并且使用它介绍两种范式，Boyce-Codd 范式(BCNF)和第三范式(3NF)。**
- **如果分解是保持依赖的，则给定一个数据库更新，所有的函数依赖都可以由单独的关系进行验证，无须计算分解后的关系的连接。**
- **我们展示了如何用函数依赖进行推导。我们着重讲了什么依赖是一个依赖集逻辑蕴涵的。我们还定义了正则覆盖的概念，它是与给定函数依赖集等价的最小的函数依赖集。**
- **我们概述了一个将关系分解成 BCNF 的算法，有一些关系不存在保持依赖的 BCNF 分解。**
- **我们用正则覆盖将关系分解成 3NF，它比 BCNF 的条件弱一些。属于 3NF 的关系也许会含有冗余，但是总存在保持依赖的 3NF 分解。**
- **我们介绍了多值依赖的概念，它指明仅用函数依赖无法指明的约束。我们用多值依赖定义了第四范式(4NF)。附录 C.1.1 给出了有关多值依赖推理的细节。**
- **其他的范式，例如 PJNF 和 DKNF，消除了更多细微形式的冗余。但是，它们难以操作而且很少使用。附录 C 给出了这些范式的详细信息。**
- **回顾本章中的要点可以发现，我们之所以可以对关系数据库设计定义严格的方法，是因为关系数据模型建立在严谨的数学基础之上。这是关系模型与我们已经学过的其他数据模型相比的主要优势之一。**

关系数据库设计的**目的**：生成一组关系模式，使我们存储信息时避免不必要的冗余，并且可以让我们可以方便地获取信息。

有损分解：分解后会损失信息：即分解之后做自然连接得到和原来不一样的表（**无损分解**：与之相反）

域是**原子的**（atomic): 该域的元素认为是不可再分的单元。

关系模式R是**第一范式**（1NF): R的所有属性的domain都是原子的。

函数依赖让我们可以表达唯一标识某些属性地值的约束。

对于**函数依赖F**,作用对象为两个属性集合 α, β

定义：$\alpha \rightarrow \beta$

具体含义为：对于该关系模式r(R)的实例中任意两元组 t1 ,t2: $t1[\alpha] = t2[\alpha] => t1[\beta] = t2[\beta]$。

有些函数依赖是**平凡的**：它们在所有的关系中都满足。

F集合的**闭包**：能够从给定F集合推导出的所有函数依赖的集合。

Botce-Codd范式（简称**BC范式**）（186页）。

着。具有函数依赖集 F 的关系模式 R 属于 BCNF 的条件是，对 F^+ 中所有形如 $\alpha \rightarrow \beta$ 的函数依赖（其中 $\alpha \subseteq R$ 且 $\beta \subseteq R$），下面至少有一项成立：

- $\alpha \rightarrow \beta$ 是平凡的函数依赖（即，$\beta \subseteq \alpha$）。
- α 是模式 R 的一个超码。

一个数据库设计属于 BCNF 的条件是，构成该设计的关系模式集中的每个模式都属于 BCNF。

保持依赖：分解关系后，函数依赖保持了下来。

第三范式（比BC范式弱）（188页）。

具有函数依赖集 F 的关系模式 R 属于**第三范式**（third normal form）的条件是：对于 F^+ 中所有形如 $\alpha \rightarrow \beta$ 的函数依赖（其中 $\alpha \subseteq R$ 且 $\beta \subseteq R$），以下至少一项成立：

- $\alpha \rightarrow \beta$ 是一个平凡的函数依赖。
- α 是 R 的一个超码。
- $\beta - \alpha$ 中的每个属性 A 都包含于 R 的一个候选码中。

注意上面的第三个条件并没有说单个候选码必须包含 $\beta - \alpha$ 中的所有属性；$\beta - \alpha$ 中的每个属性 A 可能包含于不同的候选码中。

逻辑蕴涵：

更正式地，给定关系模式 $r(R)$，如果 $r(R)$ 的每一个满足 F 的实例也满足 f，则 R 上的函数依赖 f 被 r 上的函数依赖集 F **逻辑蕴涵**（logically imply）。

属性集的闭包：

令 α 为一个属性集。我们将函数依赖集 F 下被 α 函数确定的所有属性的集合称为 F 下 α 的**闭包**，记为 α^+。图 8-8 是以伪码写的计算 α^+ 的算法。输入是函数依赖集 F 和属性集 α。输出存储在变量

3NF的优缺点：

对于关系数据库模式的两种范式——3NF 和 BCNF，3NF 的一个优点是我们总可以在满足无损并保持依赖的前提下得到 3NF 设计。然而 3NF 也有一个缺点：我们可能不得不用空值表示数据项间的某些可能有意义的联系，并且存在信息重复的问题。

第10章 存储和文件结构

- 大多数计算机系统中存在多种数据存储类型。我们可以根据访问数据的速度、购买介质时每单位数据的成本和介质的可靠性，对这些存储介质进行分类。可用的介质有高速缓冲存储器、主存储器、快闪存储器、磁盘、光盘和磁带。
- 存储介质的可靠性由两个因素决定：电源故障或系统崩溃是否导致数据丢失，存储设备发生物理故障的可能性有多大。
- 通过保留数据的多个拷贝，我们可以减少物理故障的可能性。对磁盘来说可以使用镜像技术。或者可以使用更复杂的基于独立磁盘冗余阵列（RAID）的方法。通过把数据拆分到多张磁盘上，可以提高大数据量访问的吞吐率；通过引入多张磁盘上的冗余存储，可以显著提高可靠性。有几种不同的 RAID 组织形式，它们具有不同的成本、性能和可靠性特征。最常用的是 RAID 1 级（镜像）和 RAID 5 级。
- 我们可以把一个文件从逻辑上组织成映射到磁盘块上的一个记录序列。把数据库映射到文件的一种方法是使用多个文件，每个文件只存储固定长度的记录。另一种方法是构造文件，使之能适应多种长度的记录。分槽的页方法广泛应用于在磁盘块中处理变长记录。
- 因为数据以块为单位在磁盘存储器和主存储器之间传输，所以采取用一个单独的块包含相关联的记录的方式，将文件记录分配到不同的块中是可取的。如果我们能够仅使用一次块访问就可以存取我们想要的多个记录，就能节省磁盘访问次数。因为磁盘访问通常是数据库系统性能的瓶颈，所以仔细设计块中记录的分配可以获得显著的性能提高。
- 数据字典，也称为系统目录，用于记录元数据，即关于数据的数据，例如关系名、属性名和类型、存储信息、完整性约束和用户信息。
- 减少磁盘访问数量的一种方法是在主存储器中保留尽可能多的块。因为在主存储器中保留所有的块

是不可能的，所以需要为块的存储而管理主存储器中可用空间的分配。缓冲区是主存储器的一部分，可用于存储磁盘块的拷贝。负责分配缓冲区空间的子系统称为缓冲区管理器。

前面东西操统都有；

定长记录：使用定长记录时，一般会在一个块中只分配它能完整容纳的最大的记录数（防止记录跨过块的边界）。

被删除的记录会形成一条链表，即**空闲列表**。

变长记录，变长属性——

于定长属性，如数字值、日期或定长字符串，分配存储它们的值所需的字节数。对于变长属性，如varchar类型，在记录的初始部分中表示为一个对(偏移量，长度)值，其中偏移量表示在记录中该属性的数据开始的位置，长度表示变长属性的字节长度。在记录的初始定长部分之后，这些属性的值是连

空位图的使用（258页）；**分槽的页结构**（258页）；

文件中记录组织的方法：堆文件组织、**顺序文件组织**、散列文件组织。

多表聚簇文件组织中，几个不同关系的记录存储在同一个文件中。这样的文件组织允许我们使用一次块的读操作来读取满足连接条件的记录。因此，我们可以更高效地处理这种特殊的查询。

顺序文件是为了高效处理按某个搜索码的顺序排序的记录为设计的。**搜索码**是任何一个属性或者属性的集合，不必是主码或超码。

当搜索码和物理顺序之间的一致性很低时，文件应该**重组**，使得它再一次在物理上顺序存放。

缓冲区替换策略、被钉住的块（不允许写回磁盘的块）、块的强制写出。

第11章 索引与散列

- 许多查询只涉及文件中很少一部分记录。为了减少查找这些记录的开销，我们可以为存储数据库的

文件创建索引。

- 索引顺序文件是数据库系统中最古老的索引模式之一。为了允许按搜索码顺序快速检索记录，记录按顺序存储，而无序记录链接在一起。为了允许快速的随机访问，使用了索引结构。
- 可以使用的索引类型有两种：稠密索引和稀疏索引。稠密索引对每个搜索码值都有索引项，而稀疏索引只对某些搜索码值包含索引项。 525
- 如果搜索码的排序序列和关系的排序序列相匹配，则该搜索码上的索引称为聚集索引，其他的索引称为非聚集索引或辅助索引。辅助索引可以提高不以聚集索引的搜索码作为搜索码的查询的性能。但是，辅助索引增加了修改数据库的开销。
- 索引顺序文件组织的主要缺陷是随着文件的增大，性能会下降。为了克服这个缺陷，可以使用 B^+ 树索引。
- B^+ 树索引采用平衡树的形式，即从树根到树叶的所有路径长度相等。B^+ 树的高度与以关系中的记录数 N 为底的对数成正比，其中每个非叶结点存储 N 个指针，N 值通常约为 50 ~ 100。因此，B^+ 树比其他的平衡二叉树（如 AVL 树）要矮很多，故定位记录所需的磁盘访问次数也较少。
- B^+ 树上的查询是直接而且高效的。然而，插入和删除要更复杂一些，但是仍然很有效。在 B^+ 树中，查询、插入和删除所需操作数与以关系中的记录数 N 为底的对数成正比，其中每个非叶结点存储 N 个指针。
- 可以用 B^+ 树去索引包含记录的文件，也可以用它组织文件中的记录。
- B 树索引和 B^+ 树索引类似。B 树的主要优点在于它去除了搜索码值存储中的冗余。主要缺点在于整体的复杂性以及结点大小给定时减小了扇出。在实际应用中，系统设计者几乎无一例外地倾向于使用 B^+ 树索引。
- 顺序文件组织需要一个索引结构来定位数据。相比之下，基于散列的文件组织允许我们通过计算所需记录搜索码值上的一个函数直接找出一个数据项的地址。由于设计时我们不能精确知道哪些搜索码值将存储在文件中，因此一个好的散列函数应该能均匀且随机地把搜索码值分散到各个桶中。
- 静态散列所用散列函数的桶地址集合是固定的。这样的散列函数不容易适应数据库随时间的显著增长。有几种允许修改散列函数的动态散列技术。可扩充散列是其中之一，它可以在数据库增长或缩减时通过分裂或合并桶来应付数据库大小的变化。 530
- 也可以用散列技术创建辅助索引：这样的索引称为散列索引。为使记法简便，假定散列文件组织中用于散列的搜索码上有一个隐式的散列索引。
- 像 B^+ 树和散列索引这样的有序索引可以用作涉及单个属性且基于相等条件的选择操作。当一个选择条件中涉及多个属性时，可以取多个索引中检索到的记录标识符的交。
- 对于索引属性只有少数几个不同值的情况，位图索引提供了一种非常紧凑的表达方式。位图索引的交操作相当得快，使得它成为一种支持多属性上的查询的理想方式。

基本的索引类型：**顺序索引**、**散列索引**。

搜索码：用于在文件中查找记录的属性或属性集。

索引顺序文件：在搜索码上有聚集索引的文件。

多级索引：两级或两级以上的索引。

可扩充散列和静态散列的比较：

现在让我们来看一看可扩充散列相对静态散列的优缺点。可扩充散列最主要的优点是其性能不随文件的增长而降低。此外，其空间开销是最小的。尽管桶地址表带来了额外的开销，但该表为当前前缀长度的每个散列值存放一个指针，因此该表较小。可扩充散列与其他散列形式相比，主要的空间节省是不必为将来的增长保留桶；桶的分配是动态的。

可扩充散列的一个缺点在于查找涉及一个附加的间接层，因为系统在访问桶本身之前必须先访问桶地址表。这一额外的访问只对性能有一个微小的影响。尽管 11.6 节讨论的散列结构没有这一额外的间接层，但当它们变满后就失去了这个微小的性能优势。

因而，可扩充散列看来是一种非常吸引人的技术，只要我们愿意接受在实现它时增加的复杂性。关于可扩充散列实现更详细的资料见文献注解。

（位图索引没上过吧？）

第14章 事务

- 事务是一个程序执行单位，它访问且可能更新不同的数据项。理解事务这个概念对于理解与实现数据库中的数据更新是很关键的，只有这样才能保证并发执行与各种故障不会导致数据库处于不一致状态。
- 事务具有 ACID 特性：原子性、一致性、隔离性、持久性。
  - 原子性保证事务的所有影响在数据库中要么全部反映出来，要么根本不反映；一个故障不能让数据库处于事务部分执行后的状态。
  - 一致性保证若数据库一开始是一致的，则事务(单独)执行后数据库仍处于一致状态。
  - 隔离性保证并发执行的事务相互隔离，使得每个事务感觉不到系统中其他事务的并发执行。
  - 持久性保证一旦一个事务提交后，它对数据库的改变不会丢失，即使系统可能出现故障。
- 事务的并发执行提高了事务吞吐量和系统利用率，也减少了事务等待时间。
- 计算机中不同的存储介质包括易失性存储器、非易失性存储器和稳定性存储器。易失性存储器(例如 RAM)中的数据当计算机崩溃时丢失。非易失性存储器(如磁盘)中的数据在计算机崩溃时不会丢失，但是可能会由于磁盘崩溃而丢失。稳定性存储器中的数据永远不会丢失。
- 必须支持在线访问的稳定性存储器与磁盘镜像或者其他形式的提供冗余数据存储的 RAID 接近。对于离线或归档的情况，稳定性存储器可以由存储在物理安全位置的数据的多个磁带备份所构成。
- 多个事务在数据库中并发执行时，数据的一致性可能不再维持。因此系统必须控制各并发事务之间的相互作用。
  - 由于事务是保持一致性的单元，所以事务的串行执行能保持一致性。
  - 调度捕获影响事务并发执行的关键操作，如 read 和 write 操作，而忽略事务执行的内部细节。
  - 我们要求事务集的并发执行所产生的任何调度的执行效果等价于由这些事务按某种串行顺序执行的效果。
  - 保证这个特性的系统称为保证了可串行化。
  - 存在几种不同的等价概念，从而引出了冲突可串行化与视图可串行化的概念。
- 事务并发执行所产生的调度的可串行化可以通过多种并发控制机制中的一种来加以保证。
- 给定一个调度，我们可以通过为该调度构造优先图及搜索是否无环来判定它是否冲突可串行化。然而，有更好的并发控制机制可用来保证可串行化。
- 调度必须是可恢复的，以确保：若事务 a 看到事务 b 的影响，当 b 中止时，a 也要中止。
- 调度最好是无级联的，这样不会由于一个事务的中止引起其他事务的级联中止。无级联性是通过只允许事务读取已提交数据来保证的。
- 数据库的并发控制管理部件负责处理并发控制机制。第 15 章阐述并发控制机制。

事务并非能成功执行（**中止**）、终止事务造成的变更被取消（**回滚**）、成功完成执行的事务（**已提交**）。撤销已提交事务所造成影响的唯一方法是执行一个**补偿事务**。

事务的状态：活动的、部分提交的、失败的、中止的、提交的。

事务不能正常执行后有两种做法：重启或杀死。

冲突可串行化。

冲突的定义：当I和J是不同事务在相同的数据项上的操作，并且其中至少一个是write指令时，我们说两个事务是冲突的。

如果调度S可以经过一系列非冲突的指令交换转成S',我们称S与S'**冲突等价**。

若一个调度S和一个**串行调度**冲突等价，那么称调度S是冲突可串行化的。

什么情况下，调度可冲突串行化呢——构造事务的优先图，优先图是有向图，且优先图的顶点是事务，如果事务i,j之间有边 i->j,即事务i优先事务j：满足下列三个条件：

调度的事务组成，边集由满足下列三个条件之一的边 $T_i \rightarrow T_j$ 组成：

- 在 T_j 执行 read(Q)之前，T_i 执行 write(Q)。
- 在 T_j 执行 write(Q)之前，T_i 执行 read(Q)。
- 在 T_j 执行 write(Q)之前，T_i 执行 write(Q)。

如果优先图有环，则不存在可串行化调度，否则就可以有串行化调度。

串行化顺序可以通过**拓扑排序**得到。

一个**可恢复调度**应该满足：对于每对事务T_i,T_j，如果T_j读取了之前由T_i所写的数据项，则T_i先于T_j提交。

单个事务故障导致一系列事务回滚的现象称为**级联回滚**。它会导致撤销大量工作。

无级联调度：

说，**无级联调度**（cascadeless schedule）应满足：对于每对事务 T_i 和 T_j，如果 T_j 读取了先前由 T_i 所写的数据项，则 T_i 必须在 T_j 这一读操作前提交。容易验证每一个无级联调度也都是可恢复的调度。

第15章 并发控制

锁的类型：

1. **共享的**（shared）：如果事务 T_i 获得了数据项 Q 上的**共享型锁**（shared-mode lock）（记为 S），则 T_i 可读但不能写 Q。
2. **排他的**（exclusive）：如果事务 T_i 获得了数据项 Q 上的**排他型锁**（exclusive-mode lock）（记为 X），则 T_i 既可读又可写 Q。

要访问一个数据，事务必须先给数据项加锁，申请不行，则等待。申请加锁的事务在并发管理器授权加锁之前不能执行下一个动作。

相容的：

更普遍地说，对于给定的一个锁类型集合，我们可在它们上按如下方式定义一个**相容函数**（compatibility function）：令 A 与 B 代表任意的锁类型，假设事务 T_i 请求对数据项 Q 加 A 类型锁，而事务 T_j（$T_i \neq T_j$）当前在数据项 Q 上拥有 B 类型锁。尽管数据项 Q 上存在 B 类型锁，如果事务 T_i 可以立即获得数据项 Q 上的锁，则我们就说 A 类型锁与 B 类型锁是**相容的**（compatible）。这样的一个函数可以通

事务 T_i 可以释放先前加在某个数据项上的锁。注意，一个事务只要还在访问数据项，它就必须拥有该数据项上的锁。此外，让事务在对数据项作最后一次访问后立即释放该数据项上的锁也未必是可取的，因为有可能不能保证可串行性。

授权加锁的方式：

我们可以通过按如下方式授权加锁来避免事务饿死：当事务 T_i 申请对数据项 Q 加 M 型锁时，并发控制管理器授权加锁的条件是：

1. 不存在在数据项 Q 上持有与 M 型锁冲突的锁的其他事务。
2. 不存在等待对数据项 Q 加锁且先于 T_i 申请加锁的事务。

两阶段封锁协议：保证可串行性的协议。要求每个事务分两个阶段提出加锁和解锁的申请（不保证不发生死锁。可能会级联回滚）。

- **增长阶段**：事务可以获得锁，不能释放锁
- **缩减阶段**：事务可以释放锁，但不能获得锁

两阶段封锁协议保证冲突可串行化。事务的**封锁点**：最后获得加锁的位置（增加阶段结束点）。

避免级联回滚用**严格两阶段封锁协议**：外加所有事务的**排他锁**必须在事务提交后才能释放。

强两阶段封锁协议：要求事务在提交之前不许释放任何锁。

锁转换——升级：upgrade:共享锁变排他锁、**降级**：排他锁降级为共享锁 downgrade。锁升级只发生在增长阶段，锁降级只发生在缩减阶段。

封锁的实现：用锁管理器和锁表。

基于图的协议——

所有的数据项集合D满足偏序。

偏序意味着数据集合D可视为有向无环图，称为数据库图。

树形协议——

在树形协议（tree protocol）中，可用的加锁指令只有 lock-X。每个事务 T_i 对一数据项最多能加一次锁，并且必须遵从以下规则：

1. T_i 首次加锁可以对任何数据项进行。
2. 此后，T_i 对数据项 Q 加锁的前提是 T_i 当前持有 Q 的父项上的锁。
3. 对数据项解锁可以随时进行。
4. 数据项被 T_i 加锁并解锁后，T_i 不能再对该数据项加锁。

所有满足树形协议的调度是冲突可串行化的。

树形协议保证冲突可串行化，还保证没有死锁。

缺点：

- 事务必须给那些根本不访问的数据加锁，降低并发度。
- 对一个事务集，可能不能通过树形封锁协议得到冲突可串行化调度。

死锁处理——

如果存在一个事务集，该集合中每个事务等待另一个事务，那么系统处于死锁状态。

措施：回滚死锁事务。

- 死锁预防：

排序，抢占+事务回滚、锁超时。

- 死锁检测与死锁恢复：

周期性的检测系统状态，判断有无死锁。

死锁检测

等待图，当且仅当等待图含环时，系统存在死锁。该环中每个事务都处于死锁状态。维护等待图，并周期性激活搜索环算法。

死锁中恢复的算法

- 选择牺牲者：使回滚的代价最小
- 回滚：决定事务回滚多远。彻底回滚和部分回滚。
- 饿死：把事务选为牺牲者的次数也作为代价的考虑

第16章 恢复系统

故障类型：事务故障（逻辑错误、系统错误）、系统崩溃、磁盘故障。

更新日志记录描述一次数据库写操作，有以下字段：事务标识、数据项标识、旧值、新值。

使用日志来重做和撤销事务——

恢复的过程，恢复系统使用两个恢复的过程：redo(T_i)、undo(T_i)

检查点：为了降低恢复的开销而设置。（410页）

恢复算法：**事务回滚**和**系统崩溃**后的恢复两种。（411页）